Matpar: Parallel Extensions to MATLAB

Version 1.2

Paul Springer

December 22, 1999

1. Introduction

This document describes the 1.2 release of Matpar. Matpar is a set of parallel extensions to the MATLAB program. (MATLAB is a commercial program used by scientists and engineers to perform matrix operations.) These extensions provide a much faster alternative to some of the MATLAB functions. In some cases a Matpar function is more than 30 times faster than the equivalent MATLAB function.

Part of Matpar resides on the workstation running MATLAB. This part is called the client. The other part of Matpar resides on a parallel computer, and is called the server. As of the current release, Matpar server software has been written and tested for a Beowulf computer (PC cluster running Linux). Previous versions included support for the HP/Convex Exemplar SPP2000 computer (called "neptune" at JPL/Caltech).

To make use of Matpar, the user calls a Matpar function from within MATLAB. For example, to use the Matpar QR factorization call, the user calls p_qr(), instead of the MATLAB qr() routine. Some Matpar routines have been implemented as MATLAB "M" files and some as "MEX" files. In either case, the Matpar software checks to see if this is the first such call it has received since the user began this MATLAB session. If so, Matpar uses communication software called PVM to initiate a session on the designated parallel computer. This action starts up the Matpar server software. The server software continues running until the user exits MATLAB with the "quit" command.

For each call to Matpar, the software sends the command and necessary data to the server, using PVM. The server software then executes the command, and sends back the data. The client software receives the returning data, and presents it to MATLAB in the expected format.

2. Installation

Several steps must be followed before you can run Matpar for the first time on a parallel computer. These steps are covered in this chapter. Once you follow these steps, it should never again be necessary to repeat them for that parallel computer.

2.1 Unpacking the Tar file

As of the date this document was written, the software and documentation are available for download from http://www-hpc.jpl.nasa.gov/PS/MATPAR. The appropriate files should be downloaded to the corresponding computer, and unpacked using the command "gunzip matpar1.2.server.tar.gz", followed by "tar -xvf matpar1.2.server.tar. If you are using a JPL/Caltech computer system, read the following to determine if you need to do this step on the computer you will be using.

2.2 Server Installation onto a Beowulf Parallel Computer

2.2.1 Beowulf Installation

2.2.1.1 Signing on

Matpar has been ported to a Beowulf computer running Linux. In order to use a Beowulf as a server, you must have an account you can access on the server, with a username and password. You should plan to enter your password each time to begin a session. This means you must put an "so" parameter entry into the hostfile, matpar.hosts, on the workstation side to allow for this (see section 2.3.6).

2.2.1.2 Making Matpar

Matpar requires several other software packages in order to run on the server. These include BLAS, BLACS (built on PVM, not MPI), LAPACK, ScaLAPACK, and PVM. With the possible exception of PVM, these should already be available on the parallel computer, but if not, they can be downloaded from http://netlib.org. Currently the software has only been tested with ScaLAPACK version 1.6.

If your Beowulf nodes are on a private subnet not visible from outside the cluster, Matpar requires a special version of PVM developed at JPL. This version of PVM allows the nodes to be visible to the PVM daemon running on the client. As of the writing of this document, this version of PVM has been completed, but not yet released. Release is anticipated in the very near future.

Modify the file "makefile.linux" to point to the proper locations of the libraries. Leave the DEBUG flags set as they are in the file. Then rename the file to "makefile" and do a make. This process produces an executable called "matpar".

2.2.1.3 Installing the software

PVM is responsible for initiating the server side of the software. It does this by launching a particular program that it expects to find in a particular directory. For this reason you need to make a directory called "pvm3" in your home directory. Within "pvm3" you need to make a directory called "bin" and within the "bin" directory another directory called "LINUX", or "BEOLIN" if using the special version of PVM mentioned above. Put a link to the matpar program you previously made, in this new directory. You can do so by asking your system administrator, or by entering the following commands from your home directory:

```
cd pvm3/bin/BEOLIN
ln -s /opt/local/matpar/matpar matpar
```

In the instance shown above, matpar has been previously installed in the directory "/opt/local/matpar", and the special BEOLIN version of PVM is being used.

2.2.1.4 Environment variables

Several UNIX environment variables must be set on the server side before you can use Matpar. There must be lines in the ".cshrc" file in your home directory that set these variables. See the system administrator for help on this. The lines are as follows:

```
setenv PVM_ROOT /usr/local/pvm3.4.1
```

```
setenv PVM_ARCH BEOLIN
```

The first line tells the server where the server version of PVM is. The second line is used to tell PVM what kind of server is running the program. In this example the BEOLIN version of PVM is being used. Otherwise use the keyword LINUX.

If you are using the BEOLIN version of PVM, take note of the PROC_LIST environment variable that is used to specify the names of the nodes on the machine. See the documentation that comes with this version of PVM for more information

2.3 Client Installation onto the Workstation

2.3.1 Requirements

This release, 1.2, has only been tested on SunOS 5.5 and 5.6. It must be compiled with MATLAB version 5.

2.3.2 PVM

The workstation that runs the client software must have PVM version 3.4.1 or later available to it. Furthermore, it is strongly recommended that the version of PVM to be used have its UDPMAXLEN parameter (found in the file global.h) set to 16384 or higher. If this is not done, Matpar will run extremely slowly, because of long data transmissions times. Check with the system administrator to see with what settings PVM was installed.

2.3.3 Making Matpar

Download the file matpar1.2.sun.tar.gz into the desired directory. Type “gunzip matpar1.2.sun.tar.gz” followed by “tar xvf matpar1.2.sun.tar”. Modify the makefile paths as necessary for your setup. Also modify the path for the MATLAB compiler’s cxxopts.sh file as appropriate. Make sure that the MATLAB 5 mex compiling script is in your path, and type “make”. This should compile and link the necessary modules.

2.3.4 Installing the software

If the files produced by the make command are put into the MATLAB toolbox directory, the user need not be concerned about the location of the matpar code. Otherwise, the LD_LIBRARY_PATH environment variable must be set (see below).

2.3.5 Environment variables

Several UNIX environment variables must be set on the client side before you can use Matpar. There must be lines in the “.cshrc” file in your home directory that set these variables. See the system administrator for help on this. Sample lines are as follows:

```
setenv PVM_ROOT /home/pvm/pvm_test/pvm3
setenv PVM_ARCH `$PVM_ROOT/lib/pvmgetarch`
setenv LD_LIBRARY_PATH pppp
set path = ($path $PVM_ROOT/lib)
```

The first line tells the server where the server version of PVM is. In this case a special version of PVM is invoked for MATLAB, one which transmits data back and forth more efficiently between the workstation and the server. Change this line if your version of PVM is

in a different location. The second line is used to tell PVM what kind of workstation is running the program. The third line is needed so the operating system knows from where the Matpar code should be loaded. It is not, however, needed on networks where the MATLAB startup script has been modified to set this variable correctly. If you do need this line, replace *pppp* with the name of the directory into which you put the "mpp.so" file.

2.3.6 The host file

The host file for Matpar, called matpar.hosts, is similar to the host file used by PVM. It must be located in your current directory, and is mandatory. Blank lines are permitted in the file, and any line beginning with a # character is considered to be a comment line. Each line must contain 3 fields, separated by spaces or tabs. The first field contains the name of a parallel computer to which a connection can be made. The second field may itself contain spaces, and comprises the parameters passed to PVM. The parameters are separated by spaces within the field.

The main parameters of interest are the "so" parameter and the "dx" parameter. The "so" parameter is used to indicate that no ".rhosts" file is on the parallel computer, and that therefore a password is required. The parameter takes the form "so=pw".

The "dx" parameter is required, and tells the program where to find pvmd, the PVM daemon, on the server. It takes the form "dx=/usr/bin/pvmd" where the path shown should be replaced by whatever path is appropriate. When individual script files are needed for each processor configuration, an entry for each such configuration must appear in the hosts file, as shown below.

The third field tells how many nodes. This is only useful for computers which need different startup scripts for differing numbers of nodes requested. In that situation, the dx parameter differs for each different node size request. For computers where this does not matter, put –1 in this field.

A sample matpar.hosts file follows.

```
# matpar.hosts file for the SPP2000 "neptune"

# 4 entries for the SPP2000, for 4, 16, 32 and 64 processors

neptune    dx=/home/myname/pvm3/bin/CSPP/pvmdsz4          4
neptune    dx=/home/myname/pvm3/bin/CSPP /pvmdsz16        16
neptune    dx=/home/myname/pvm3/bin/CSPP /pvmdsz32        32
neptune    dx=/home/myname/pvm3/bin/CSPP /pvmdsz64        64
```

3. Running with Matpar

The Matpar code does not run until one of its calls (beginning with the characters "p_") is invoked. Some time before such a call is made, you must have begun the PVM daemon on the workstation, called pvmd. Start the daemon by opening a new window on your workstation, and typing "pvmd". Make sure this is done in a separate window, and is not run in the background, so that you can be prompted for a password (if necessary) in the pvmd window. If the system complains that it can't find pvmd, check your path to be sure that it includes the proper PVM directory (see section 2.3.5).

The first Matpar command other than p_config() will initiate the session on the parallel computer. The Matpar commands can be typed directly in to MATLAB for immediate execution, or they may be part of a MATLAB program file. When the first command is executed, Matpar will print out certain status messages. An example is shown below:

```
Initiating Matpar 1.2
Adding host neptune.cacr.caltech.edu to virtual machine...
Starting 4 matpar tasks
in 2 rows and 2 columns
```

This message shows which parallel computer is being used, how many nodes have been requested, and what configuration the nodes are in. This step can take some time, if many nodes have been requested, or if someone else is using the machine, and your request must wait its turn.

When Matpar sends a matrix to the server, you will see a message like the following:

```
Sending to 524289
```

If the matrix is so large that is must be broken up into pieces in order to send it efficiently, Matpar will display the following message:

```
Sending matrix block
```

If the result from the server is also too large, you will see this message:

```
Unpacking block
```

When you exit your MATLAB session by typing “quit”, this will also terminate the server part of Matpar running on the parallel computer, and will terminate the pvmd program that you began as part of the run. For that reason pvmd must be restarted when you begin another MATLAB session which uses Matpar.

If there is a problem on the server computer that causes the Matpar server code to abort, the next time you run Matpar you may see the message

```
No tasks started because of Duplicate Host error
May have to remove /tmp/pvmd.xxx file on remote system
```

This indicates that PVM on the server did not exit cleanly, and left a temporary file that must be deleted before you can run. The file to be deleted is in the directory “/tmp” and is called “pvmd.xxx”, where “xxx” represents your user ID on the server.

4. Constraints

Matpar users are constrained by the same restrictions affecting other users of the parallel computers. These may include restrictions imposed by the scheduling software for that system, such as how long a job may run, and whether it has to wait for other jobs to complete before it starts.

Memory constraints impose limitations on the size of matrices that can be handled. For example, on an SPP2000 with 4 GB of memory per hypernode, a matrix larger than 6000 x 6000 elements may be too large. For the Bode plot calculations, the size of the state matrix must be even smaller--perhaps as small as 2000 x 2000 elements. Most computers will not crash if these limits are exceeded, but will begin using virtual memory, and will run very slowly.

ScaLAPACK, a set of library routines that Matpar uses for parallel matrix calculations, requires that the data be distributed across the nodes in certain ways, and this can vary depending on the operation being performed. When a matrix is passed to the server as part of a Matpar calculation, Matpar distributes the data correctly for that operation. However, if the data is initially passed as a persistent matrix, or if a result from a previous operation is made persistent and used in a succeeding operation, the data may not be distributed correctly. Incorrect data distribution will produce an error, but this can only happen if persistence is used. Once the new version of ScaLAPACK is integrated into Matpar, this problem will be corrected.

If the computer on which your data resides does not have a high speed Ethernet connection to the Internet, your performance times will suffer. In particular, the Ethernet card should be capable of speeds of at least 100 megabits per second. UltraSparc computers come with these high-speed cards as part of their normal configuration.

Matpar can not process sparse matrices in the current release. Any sparse matrices you want Matpar to handle must be passed via the MATLAB **full()** command, or an error will result.

The matrices are assumed to hold real double precision values. A few operations allow for complex double precision values to be used, and these are noted below.

5. Matpar Calls

5.1 Persistence

When a matrix is available for use beyond the immediate operation for which it was sent to the server, that matrix is called *persistent*. The user of persistence can greatly enhance the performance of Matpar. If, for example, matrix A is used in 10 Matpar operations, it need be transferred only once if it is made persistent. If A is a large matrix this can result in a big savings in communication time.

A matrix may be declared persistent using the p_persist() command, or the result of a Matpar operation may be made persistent, using the optional **result** parameter. To save space, when the persistent matrix is no longer needed it can be deleted with the p_delete() command. A persistent matrix is referenced by an integer between 1 and 10.

Square brackets [ ] in the following calls denote parameters that are optional.

p_config(machine, size)

Purpose: Specify which parallel computer and how many nodes will be used for this MATLAB session. In the current release, if this call is not made, Matpar uses 4 nodes on “grand-canyon”. This call my not be made to stop one parallel session and start another, though that may be allowed in a future release. You are currently limited to a single invocation of this call in any MATLAB session.

Parameters:

machine: name of the parallel computer on which to open a session, contained in single quotes.

size: number of nodes to use on the parallel computer
returns: nothing

Example:
p_config('cosmos', 32) -- Use 32 nodes on the computer *cosmos.*

p_add(A, B[, result, consumeFlag])

p_sub(A, B[, result, consumeFlag])

Purpose: Add (or subtract) matrices A and B

Parameters:
A: one of the matrices to add (subtract). scalar value implies a reference to a persistent matrix.
B: the other matrix to add (subtract). scalar value implies a reference to a persistent matrix.
result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.
consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.
returns: nothing if **result** is present.
otherwise the result of the matrix-matrix operation.

MATLAB equivalent: A + B, A - B

Examples:
C = p_add(A, B)
p_sub(A, 1, 2, 'N') -- Subtract persistent matrix 1 from A. Save the result on the parallel computer as a persistent matrix, to be referenced by the number 2.

p_bode(A, B, C, D, w)

Purpose: Generate frequency response output matrix

Parameters:
A: state matrix for the system; may be complex
B: system input vector / matrix
C: system output vector / matrix
D: offsets to be added in to the final result
w: frequency vector; may be complex
returns:
mag = magnitude of combined frequency response matrix for all columns in **B**.

phase = phase of combined frequency response matrix for all columns in **B**.
wOut = same as **w**

MATLAB equivalent: bode(a, b, c, d, iu, w)), provided B has 1 column. . If B has more than one column, Matpar combines the information for all columns of B into a single aggregate matrix.

Example:

```
A = rand( 100, 100 )
B = rand( 100, 1 )
C = rand( 1, 100 )
D = zeros( 1 )
w = rand( 1, 100 ) + i * rand( 1, 100 )
[mag, phase, wOut] = p_bode( A, B, C, D, w )
```

p_delete(matnum)

Purpose: Delete a persistent matrix residing on the parallel computer

Parameters:

matnum: the reference number of the persistent matrix to be deleted
returns: nothing

Example:

```
p_persist( A, 1 )        % make matrix A persistent
p_delete( 1 )            % now delete it
```

p_eye(m[, n], result, consumeFlag)

Purpose: Generate identity matrix on the server

Parameters:

m: number of rows in the identity matrix. If **n** is not specified, **m** also indicates the number of columns.
n (optional): number of columns in the identity matrix
result: Store the result on the parallel computer as a persistent matrix to be referenced in the future by **result**.
consumeFlag: If set to 'C', delete this identity matrix immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.
returns: Identity matrix is generated as a persistent matrix on the parallel computer. Nothing is returned.

MATLAB equivalent: eye(m[,n])

Example:

p_eye(40, 5, 'c') -- Generate an identity matrix of order 40. Save it as persistent matrix 5, to be consumed the next time it is referenced.

p_freqresp(A, B, C, D, w)

Purpose: Generate frequency response output matrix

Parameters:

A: transfer matrix for the system; may be complex
B: system input vector
C: system output vector
D: offsets to be added in to the final result
w: frequency vector; may be complex
returns: combined frequency response matrix for all columns of **B**.

MATLAB equivalent: freqresp(A, B, C, D, 1, w), provided B has 1 column. . If B has more than one column, Matpar combines the information for all columns of B into a single aggregate matrix.

Example:

```
A = rand( 100, 100 )
B = rand( 100, 1 )
C = rand( 1, 100 )
D = zeros( 1 )
w = rand( 1, 100 ) + i * rand( 1, 100 )
G = p_freqresp( A, B, C, D, sqrt(-1) * w )
```

p_inv(A[, result, consumeFlag])

Purpose: Compute the inverse for the square matrix A

Parameters:

A: the matrix to be inverted. A scalar value implies a reference to a persistent matrix
result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.
consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.
returns: nothing if **result** is present.
Otherwise it returns the inverted matrix.

MATLAB equivalent: inv(A)

Example:

```
A = rand( 4, 4 )
B = p_inv( A )
```

p_lu(A[, result, consumeFlag])

Purpose: Compute LU factorization for the matrix A

Parameters:

A: the matrix to be factored. scalar value implies a reference to a persistent matrix

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present.
Otherwise it returns the factorization matrix, with U in the upper triangular part, and L (without its unit diagonal elements) stored in the lower triangular.

MATLAB equivalent: lu(A)

Example:

```
A = rand( 4, 4 )
B = p_lu( A )
```

p_mult(A[, 'T'] B[, 'T'][, result, consumeFlag])

Purpose: Multiply matrix A (or its transpose) times matrix B (or its transpose)

Parameters:

A: one of the matrices to multiply. scalar value implies a reference to a persistent matrix.

B: the other matrix to multiply. scalar value implies a reference to a persistent matrix.

'T': If the letter 'T' appears after either A or B (or both), use the transpose.

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present.
otherwise the result of the matrix-matrix multiplication.

MATLAB equivalent: A * B

Examples:

M = p_mult(A, B)

p_mult(A, 1, 2, 'N') -- Multiply A times persistent matrix 1. Save the result on the parallel computer as a persistent matrix, to be referenced by the number 2.

p_multtrans(A, transFlag[, result, consumeFlag])

Purpose: Multiply matrix A times its transpose

Parameters:

A: the matrix to multiply. scalar value implies a reference to a persistent matrix.

transFlag: when set to 'T', multiply A' * A. If set to 'N' or any other letter, multiply A * A'

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present. otherwise the result of the matrix-matrix multiplication.

MATLAB equivalent: A * A'

Examples:

M = p_multtrans(A, 'T')

p_multtrans(1, 'N', 2, 'N') -- Multiply persistent matrix 1 times its transpose. Save the result on the parallel computer as a persistent matrix, to be referenced by the number "2".

p_persist(A, matnum[, consumeFlag])

Purpose: Save matrix A as a persistent matrix on the parallel machine, to be referenced by **matnum**.

Parameters:

A: the matrix to be made persistent

matnum: the number by which to refer to this matrix in future operations

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing

Example:

A = rand(1000, 1000)

p_persist(A, 1, 'C') -- Make A a persistent matrix. Delete the persistent matrix after its next use. The next time it is referenced on the parallel machine, the number 1 will be its reference number.

p_pinv(A[, tol][, result, consumeFlag])

Purpose: Compute the pseudoinverse for the matrix A

Parameters:

A: the matrix to be inverted. A scalar value implies a reference to a persistent matrix

tol (optional): the tolerance value to use. The algorithm uses singular value decomposition, and any singular values less than **tol** are considered to be 0. Default tolerance is the same one used by MATLAB.

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present.
Otherwise it returns the inverted matrix.

MATLAB equivalent: pinv(A)

Example:

A = rand(4, 7)

p_inv(A, 8, 'k') -- Generate pseudoinverse of matrix A. Save it as a persistent matrix to be referenced by the number '8'.

p_qr (B[, rsize][, result, consumeFlag])

Purpose: Perform QR factorization on B, returning the R matrix.

Parameters:

B: the matrix to factorize. scalar value implies a reference to a persistent matrix

rsize (optional): how much of the lower right quadrant of R to return. If rsize is 1, return the element in the highest numbered row and column. If this parameter is missing, rsize is set equal to the smallest dimension of B. If B is square, this will return all of R.

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present, otherwise the matrix R--ignore the values in R below the diagonal

MATLAB equivalent: triu(p_qr(B) = triu(qr(B))

Example:

```
B = rand( 4, 4 )
R = p_qr( B, 4 )
```

p_smult(s, A[, result, consumeFlag])

Purpose: Multiply matrix A by the scalar s.

Parameters:

s: the scalar by which to multiply matrix A

A: the matrix to be multiplied. A scalar value implies a reference to a persistent matrix

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present.
Otherwise it returns the multiplied matrix.

MATLAB equivalent: s * A

Example:

```
p_eye( 16, 9, 'c' )       % generate identity matrix of order 16
p_smult( 8, 9, 4, 'k' )   % make persistent matrix 4 with 8's on diagonal
```

p_solve(A, B[, result, consumeFlag])

Purpose: Solve the equation AX = B, returning X

Parameters:

A: the matrix on the left side of the equation. scalar value implies a reference to a persistent matrix.

B: the matrix on the right side of the equation. scalar value implies a reference to a persistent matrix.

result (optional): Do not return the result. Instead store it on the parallel computer as a persistent matrix to be referenced in the future by **result**.

consumeFlag (optional): If set to 'C', delete the persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** is present. otherwise the matrix X.

MATLAB equivalent: A \ B

Example:

```
A = rand( 4, 4 )
B = rand( 4, 4 )
X = p_solve( A, B )
```

[S =]p_svd(A[, result, consumeFlag])

[U, S, V =]p_svd(A, 0[, Uresult, UconsumeFlag, Sresult, SconsumeFlag, Vresult, VconsumeFlag])

Purpose: Compute the singular value decomposition for the matrix A. In the first case, return only the singular values contained in the vector S, in decreasing order. In the second case, return unitary matrices U and V, and diagonal matrix S containing the singular values in decreasing order. In this case, only the "economy size" decomposition is used.

Parameters:

A: the matrix for which to determine the singular values. A scalar value implies a reference to a persistent matrix.

S: the vector / matrix containing the singular values

U, V: the unitary matrices returned. A = U * S * V'.

Uresult, Sresult, Vresult (optional): Do not return the results. Instead store them on the parallel computer as a persistent matrices to be referenced in the future by **Uresult, Sresult,** and **Vresult**.

UconsumeFlag, SconsumeFlag, VconsumeFlag (optional): If set to 'C', delete the corresponding persistent matrix result of this operation immediately after the next time it is referenced. If set to 'K' or any other letter, the matrix is kept on as a persistent matrix.

returns: nothing if **result** parameters are present.
Otherwise it returns the singular value vector(first form) or the singular value matrix **S** and the unitary matrices **U** and **V**.

MATLAB equivalent:
S = svd(A)
[U, S, V] = svd(A, 0)

Example:
A = rand(4, 4)
p_svd(A,0, 1, 'c', 2, 'k', 3, 'c') -- Calculate singular value decomposition of A. Store it in persistent matrix 2, and save U in matrix 1 and V in matrix 3. U and V should not be kept beyond their next reference.

p_trace(A, transFlag)

Purpose: Compute trace of A * A^T or A^T * A.

Parameters:

A: the matrix for the computation. scalar value implies a reference to a persistent matrix

transFlag: If set to 'T', calculate trace of A^T * A. If set to 'N', calculate trace of A * A^T

returns: trace of the result

MATLAB equivalent: trace(A * A')

Example:
A = rand(5, 4)
x = p_trace(A, 'N')